Programming Languages Are Like Cars

If we use cars as an analogues for languages, then BASIC is the family sedan. It's inexpensive, there are lots of models in the range, and the most popular models have all kinds of (previously optional) extras fitted as standard.

Ah yes, **Pascal**. I only ever wrote one major package in Pascal, and would never use it again, if I could avoid it. Pascal is OK for academia, where all the people who will use a program are computer literate, familiar with the operating system, and so on. But in the real world, programs get used by office juniors, managing directors and other people who are completely fazed by error messages and who are liable to type the most improbable input string imaginable.

Real-world programs, therefore, have to have extensive input validation, and this turns out to be quite difficult in Pascal, as it is tediously uncooperative in type conversions. The Pascal programmer has thus to resort to all kinds of devious practices to get Pascal to work for him, which defeats the purpose of using the language in the first place. If you have to use sleight of hand so that the compiler can't understand what you're doing, there's a fair chance you can't understand it yourself.

The right place for Pascal, then, is where you have to write programs of moderate sophistication which you will only ever use yourself, and where you are sure that a straight solution will do. I know this is cruel and a bit over the top, but the best analogy I can find for Pascal is a trainer bicycle. You can't go far with it, you can't go fast with it, you can't carry much with it, it's not to everybody's taste, but at least you can't fall over while riding it.

FORTRAN lives on in the scientific world. First of all, most engineers and scientists cut their teeth on it, so that it is widely known. Secondly, it is one of the first languages every mainframe and mini manufacturer provides on their machines, so that programs originally written on larger computers can be transported to micros and vice versa. The fact that it is well standardised assists this. A third factor is the vast amount of FORTRAN software which has been published or placed in the public domain for all kinds of purposes.

The sheer inertia of the FORTRAN movement ensures that it will be around for a long time, and a FORTRAN compiler is a must for every micro in a scientific or engineering lab. However, that's not to say that FORTRAN is always the best tool for the job. Pascal can be appropriate here for simple jobs, or PL/I (with its superb double-precision arithmetic and hyperbolic functions) for the more complex ones. C is also an appropriate tool on occasions. However, those who own FORTRAN compilers should bear in mind that Real Programmers use FORTRAN.

FORTRAN is a bit like a Volkswagen Beetle; the design hasn't changed that much over the years, and people wouldn't like it so much if it had. It's a very rational and appropriate design, with few concessions to style or fashion.

COBOL finds its metier in the commercial world where programs have long lives and may be worked on by as many as ten programmers during that time. Consequently the requirement is not for flash tricky code, but for code that your average commercial programmer can pick up, understand and modify or fix. While the previous generation of COBOL compilers acquired a reputation for being as slow as a wet weekend, the latest releases are much improved, making COBOL a viable alternative for the commercial micro user or software house.

COBOL is like a half-ton truck: it serves the needs of commerce with little style. It can carry quite heavy loads, but will never turn heads as it passes.

PL/I - ah, now here's a language for Real Programmers. I turned to PL/I after my nasty experience with Pascal and have never regretted it. Block structures and structured programming statements like Pascal's, combined with sophisticated file handling and I/O and the ability to do binary arithmetic for scientific applications and decimal arithmetic for dollars and cents mean that here is a language that one can stick with. While the rest of the world follows fashions like Pascal and C, we PL/I programmers will be quietly getting on with the job.

If you're a COBOL programmer who wants to have some good structured programming support and less verbose code, then PL/I may be for you. If you're a FORTRAN programmer who wants higher precision arithmetic, better string handling and structured programming support, PL/I is it. If you're a Pascal programmer who feels the need for the occasional goto - used with discretion, of course - as well as sensible file handling, check out PL/I. If you're a C programmer, nothing I say is going to change your mind anyway; but PL/I has pointers, structures, unions, functions, storage classes and all the other things that make C such fun.

PL/I therefore emerges as a good all-round language with particular strengths for commercial software and scientific, though utilities compiled in it tend to be a bit large. It's a bit like these new family vans with seats that fold, turn and twist: you can use it as a sedan, as a minibus, as a delivery van or as a camper. Bear in mind that unlike those vans, it has a V12 under the bonnet.

C is a sports car in comparison. It's small, zippy and manoeuverable, very light in weight and when it crashes, you're a gonner (due to the lack of run-time debugging facilities, you see). C is best used to write systems utilities such as archivers, macro processors, editors, compilers and the like. It can be used to write commercial or scientific programs, though its arithmetic and file handling let it down in the former area. If you build on its lower-level functions to construct nice string-handling and I/O functions, what you wind up with is very like PL/I, which would have made a more appropriate starting point.

The best feature of C is its portability. C programs can generally be moved from system to system with the minimum of effort, and this means that software authors can be assured of achieving the maximum return for their effort.

Prolog is quite different from most languages. At the moment, it's still barely out of the research labs, and most users in Australia are in Universities. I dare say that Prolog could be put to commercial use, particularly in bibliographic databases, small 'expert' systems and the like.

In the car metaphor, Prolog is rather like those experimental designs that the manufacturers roll out every now and again, informing us that in ten years time, we'll all be driving cars like these, with four-wheel steering and the like.

Ada is an armoured personnel carrier. It's designed to be reliable, carry all kinds of loads, be fast and it has a radio built in so it can communicate with other carriers.

Ada supports a number of recent structured programming concepts directly. For example, the idea of separate compilation of modules has been around for a long time; many Pascal, C and PL/I compilers offer this facility. But Ada is the only language to date to support the separate compilation of modules (Ada calls them packages), as part of the language.

Ada actually addresses a number of problems which have emerged as recent issues in software engineering. For example, we now know that the cost of software is significantly higher than that of hardware. We know that the cost of fixing bugs increases rapidly the later they are discovered, and that the cost of maintaining a program is high in comparison with the original cost of writing it.

Excerpts from: Les Bell, Languages and Software Development, 1985